Manipulação de Vetores

Loana Tito Nogueira
loana@id.uff.br

Operações sobre vetores

- Vetores são normalmente chamados de Lista em Python
- Muito do que vocês aprenderam a fazer já está pronto como funções do Python
  - Contar o número de vezes que um determinado elemento aparece dentro de uma lista
  - Inserir um elemento no final da lista
  - Inserir um elemento em uma determinada posição
  - Concatenar duas listas
  - Excluir um elemento que está numa determinada posição, ou que tem um determinado valor
  - Encontrar a posição de um elemento dentro de uma lista
  - Ordenar uma lista
  - Saber se um determinado elemento está em uma lista

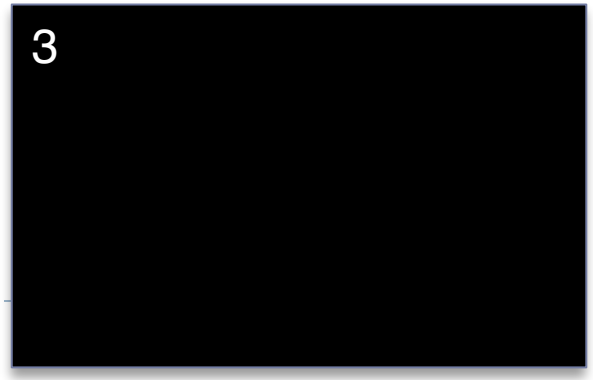
Contar Elementos

- Já sabemos contar elementos
- Exemplo: contar quantas vezes o elemento de valor 10 aparece na lista

Contar Elementos

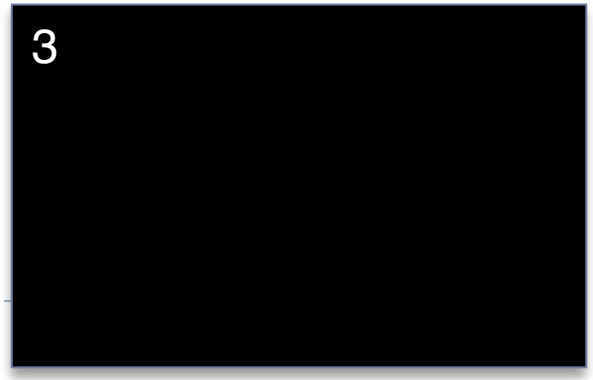
- Já sabemos contar elementos
- Exemplo: contar quantas vezes o elemento de valor 10 aparece na lista

```
lista = [1, 10, 2, 10, 3, 10, 4, 5, 6]
cont = 0
for i in range(len(lista)):
    if lista[i] == 10:
        cont += 1
print(cont)
```



Alternativa: método **count(elemento)**

```
lista = [1, 10, 2, 10, 3, 10, 4, 5, 6]
cont = lista.count(10)
print(cont)
```



Inserir um elemento no final da lista

- Já sabemos usar o método **append(elemento)**

```
>>> lista = [1, 2, 3]
>>> lista.append(4)
>>> lista
[1, 2, 3, 4]
```

Concatenar duas listas

- Inserir os elementos de lista2 ao final de lista1

Concatenar duas listas

- Inserir os elementos de lista2 ao final de lista1

```
lista1 = [1, 2, 3, 4]
lista2 = [5, 6, 7]
for i in range(len(lista2)):
    lista1.append(lista2[i])
print(lista1)
```

```
[1, 2, 3, 4, 5, 6, 7]
```

Alternativa: +

- Usar a operação de soma (+)

```
lista1 = [1, 2, 3, 4]
lista2 = [5, 6, 7]
lista1 = lista1 + lista2
print(lista1)
```

```
[1, 2, 3, 4, 5, 6, 7]
```

Alternativa: método **extend(lista2)**

- Inserir os elementos de lista2 ao final de lista1

```
lista1 = [1, 2, 3, 4]
lista2 = [5, 6, 7]
lista1.extend(lista2)
print(lista1)
```

```
[1, 2, 3, 4, 5, 6, 7]
```

Inserir um elemento em uma determinada posição

- Inserir o elemento 10 na posição 2 de uma lista (ou seja, inserir como terceiro elemento da lista)

Inserir um elemento em uma determinada posição

- Inserir o elemento 10 na posição 2 de uma lista (ou seja, inserir como terceiro elemento da lista)

```
lista = [1, 2, 3, 4]
lista.append(0)
pos = 2
#abre espaço para inserir o novo elemento
for i in range(len(lista)-1, pos-1, -1):
    lista[i] = lista[i-1]
lista[pos] = 10
print(lista)
```

```
[1, 2, 10, 3, 4]
```

Alternativa: método **insert(indice, elemento)**

```
lista = [1, 2, 3, 4]
lista.insert(2, 10)
print(lista)
```

```
[1, 2, 10, 3, 4]
```

Excluir um elemento

- Excluir o elemento da posição 2 da lista [1, 2, 3, 4]

Excluir um elemento

- Excluir o elemento da posição 2 da lista [1, 2, 3, 4]

```
lista = [1, 2, 3, 4]
pos = 2
temp = []
for i in range(len(lista)):
    if i != pos:
        temp.append(lista[i])
lista = temp
print(lista)
```

```
[1, 2, 4]
```

Alternativa: método **pop(índice)**

```
lista = [1, 2, 3, 4]
lista.pop(2)
print(lista)
```

```
[1, 2, 4]
```

Método **pop()**

- Quando o índice é omitido, o método pop remove o último elemento da lista

```
lista = [1, 2, 3, 4]
lista.pop()
print(lista)
```

```
[1, 2, 3]
```

Excluir um elemento

- Excluir a primeira ocorrência do elemento de valor 4 da lista [1, 4, 5, 6, 4, 7]

Excluir um elemento

- Excluir a primeira ocorrência do elemento de valor 4 da lista [1, 4, 5, 6, 4, 7]

```
lista = [1, 4, 5, 6, 4, 7]
valor = 4
removeu = False
temp = []
for i in range(len(lista)):
    if lista[i] != valor or removeu:
        temp.append(lista[i])
    else:
        removeu = True
lista = temp
print(lista)
```

```
[1, 5, 6, 4, 7]
```

Alternativa: método **remove(elemento)**

```
lista = [1, 4, 5, 6, 4, 7]
lista.remove(4)
print(lista)
```

```
[1, 5, 6, 4, 7]
```

Encontrar um elemento dentro da lista

- Encontrar a primeira ocorrência do elemento de valor 10 na lista [1, 2, 10, 5, 20] e retornar a posição em que ele foi encontrado

Encontrar um elemento dentro da lista

- Encontrar a primeira ocorrência do elemento de valor 10 na lista [1, 2, 10, 5, 20] e retornar a posição em que ele foi encontrado

```
lista = [1, 2, 10, 5, 20]
valor = 10
pos = -1
for i in range(len(lista)-1,-1,-1):
    if lista[i] == valor:
        pos = i
print(pos)
```

```
2
```

Alternativa: index(elemento)

```
lista = [1, 2, 10, 5, 20]
pos = lista.index(10)
print(pos)
```

```
2
```

Método **index(elemento)**

- Quando o elemento procurado não está na lista, o método index lança uma exceção

```
>>> lista = [1, 2, 3, 4]
>>> lista.index(7)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 7 is not in list
```

Teste de Pertinência

- Retornar True caso o valor 10 pertença à lista, e False caso contrário

Teste de Pertinência

- Retornar True caso o valor 10 pertença à lista, e False caso contrário

```
lista = [1, 2, 3, 4]
valor = 7
resultado = False
for i in range(len(lista)):
    if lista[i] == valor:
        resultado = True
print(resultado)
```

False

Alternativa: **elemento in lista**

```
lista = [1, 2, 3, 4]
resultado = 7 in lista
print(resultado)
```

False

Ordenar uma lista

- Selection Sort, Insertion Sort, Bubble Sort, etc

Ordenar uma lista

[1, 2, 3, 3, 4, 5, 7, 8, 9, 10, 11]

- Selection Sort

```
lista = [10, 9, 8, 7, 5, 3, 4, 3, 1, 2, 11]
lista_ordenada = []
for i in range(len(lista)):
    menor = lista[0]
    for j in range(len(lista)):
        if lista[j] < menor:
            menor = lista[j]
    lista_ordenada.append(menor)
    lista.remove(menor)
print(lista_ordenada)
```

Alternativa: método **sort**()

```
lista = [10, 9, 8, 7, 5, 3, 4, 3, 1, 2, 11]
lista.sort()
print(lista)
```

```
[1, 2, 3, 3, 4, 5, 7, 8, 9, 10, 11]
```

Ordem decrescente

- Também é possível inverter a lista usando **reverse()**

```
lista = [1, 3, 2, 4]
lista.reverse()
print(lista)
```

[4, 2, 3, 1]

Fatias (*Slices*)

- Até agora acessamos os elementos de uma lista usando índices únicos
- Os elementos de uma lista também podem ser acessados por faixas de índice
  - Em Python, isso é feito usando a notação de fatias (*slices*)

Slices

- lista[inicio:fim]
  - Retorna a lista formada pelo elemento que está na posição **início** até o elemento que está na posição **fim – 1**
- lista[:fim]
  - Retorna a lista formada pelo elemento que está na posição **0** até o elemento que está na posição **fim – 1**
- lista[inicio:]
  - Retorna a lista formada pelo elemento que está na posição **início** até o último elemento
- lista[:]
  - Retorna a lista toda

Exemplos

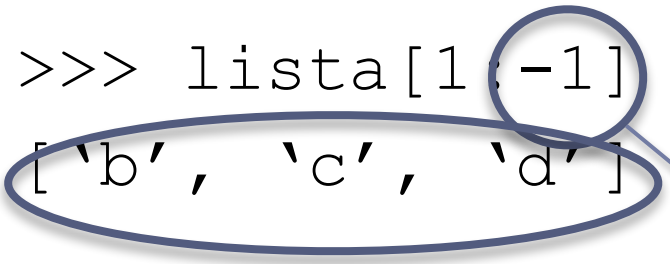
```
>>> lista = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
>>> lista[2:]
[‘c’, ‘d’, ‘e’]

>>> lista [:3]
[‘a’, ‘b’, ‘c’]
```

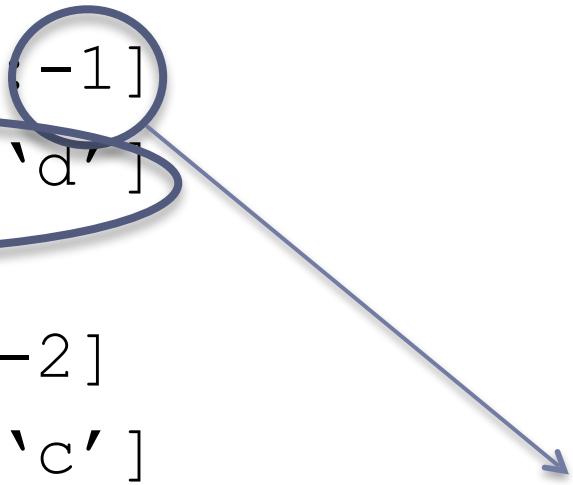
Exemplos

```
>>> lista = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’]
>>> lista[1:3]
[‘b’, ‘c’]

>>> lista[1:-1]
[‘b’, ‘c’, ‘d’]

>>> lista[:-2]
[‘a’, ‘b’, ‘c’]
```

Exemplos

```
>>> lista = ['a', 'b', 'c', 'd', 'e']
>>> lista[1:3]
['b', 'c']

>>> lista[1:-1]
['b', 'c', 'd']

>>> lista[:-2]
['a', 'b', 'c']
```





Notar que o slice vai até fim – 1, portanto esse comando retorna até a penúltima posição

Slice com Incremento

- lista[inicio:fim:passo]
  - Retorna os elementos que vão da posição **início** até a posição **fim-1**, com incremento **passo**

Exemplos

```
>>> lista = [1, 2, 3, 4, 5, 6]
>>> lista[:-1:2]
[1, 3, 5]
```

Atribuição

- Já sabemos que elementos de uma lista podem ser substituídos por outro elemento qualquer – basta usar o índice da posição do elemento que queremos substituir

```
>>> lista = [1, 2, 3, 4, 5, 6]
>>> lista[3] = 'a'
>>> lista[5] = 'b'
>>> lista
[1, 2, 3, 'a', 5, 'b']
```

Atribuição com slices

- Quando se usa *slices* numa atribuição, os elementos que estão na fatia são substituídos

```
>>> lista = [1,2,3,4,5]
>>> lista[1:3] = ['a', 'b']
>>> lista
[1, 'a', 'b', 4, 5]
```

Atribuição com *slices*

- Quando se usa *slices* numa atribuição, os elementos que estão na fatia são substituídos

```
>>> lista = [1,2,3,4,5]
>>> lista[1:3] = ['a', 'b']
>>> lista
[1, 'a', 'b', 4, 5]
```

Equivale a

```
>>> lista[1:3] = 'a', 'b'
```

Exceto quando a lista da direita tem apenas um elemen
Nesse caso ela deve estar envolta em []

Atribuição com *slices*

- O que acontece quando se atribui uma quantidade de elementos menor (ou maior) do que a fatia determinada no comando?
  - A atribuição substitui os elementos da fatia pelos novos elementos, **independente da quantidade**

```
>>> lista = [1, 2, 3, 4, 5]
>>> lista[1:3] = [10, 20, 30, 40]
>>> lista
[1, 10, 20, 30, 40, 4, 5]
```

Atribuição com *slices*

- O que acontece quando a fatia especificada no comando de atribuição é **maior** que a quantidade de elementos da lista que está sendo atribuída?
  - O comando de atribuição substitui os elementos do início da fatia até o fim da lista pelos novos elementos, estendendo a lista, se necessário

Exemplo

```
>>> lista = [1, 2, 3, 4, 5, 6]
>>> lista[2:10] = [7]
>>> lista
[1, 2, 7]

>>> lista = [1, 2, 3, 4, 5, 6]
>>> lista[1:10] = [7, 8, 9, 10, 11, 12]
>>> lista
[1, 7, 8, 9, 10, 11, 12]
```

Atribuição com *slices*

- Exceção: quando usamos **incremento**, a quantidade de elementos inseridos e o tamanho da fatia devem ser compatíveis, caso contrário o Python gerará um erro

Exemplo sem erro

```
>>> lista = [1, 2, 3, 4, 5]
>>> lista[::2] = ['x', 'y', 'z']
>>> lista
['x', 2, 'y', 4, 'z']
```

Exemplo com erro

```
>>> lista = [1, 2, 3, 4, 5]
>>> lista[0::2] = [6, 7]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: attempt to assign sequence of
size 2 to extended slice of size 3
```

Representação de Listas em Memória

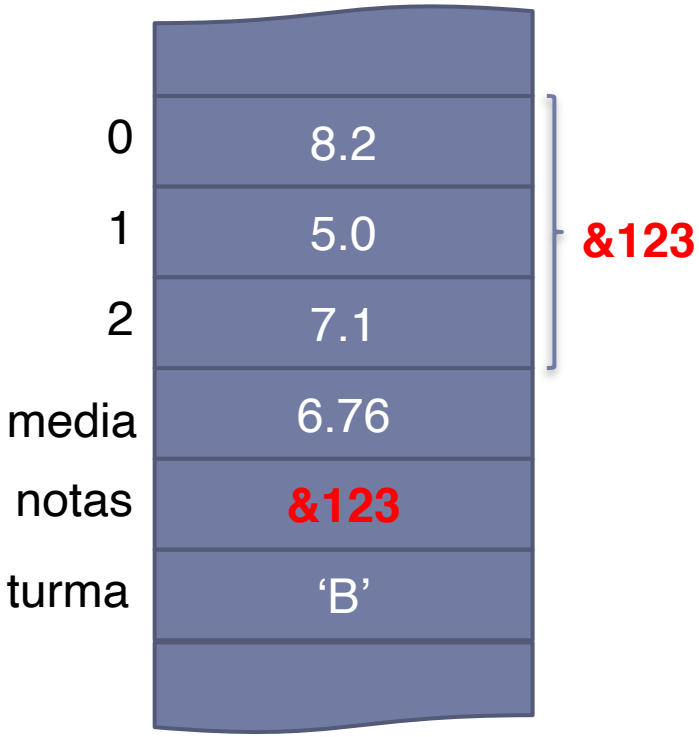
- O valor de uma variável de lista na verdade é um endereço de memória

Representação de Listas em Memória

Em Phyton

```
notas = [8.2, 5.0, 7.1]
turma = 'B'
media = 0
for i in range(len(notas)):
    media = media + notas[i]
media = media/len(notas)
```

Na Memória



Cópia de listas

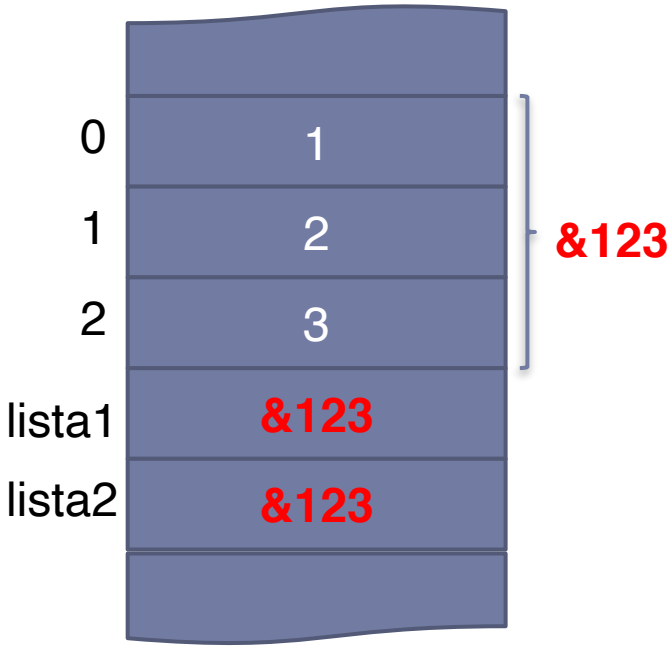
- Ao copiar uma lista para outra, o que é feito é copiar o valor do endereço de memória
  - Ambas passam a apontar para o mesmo endereço, portanto o que for modificado em uma lista também será modificado na outra

Cópia de Listas

Em Phyton

```
>>> lista1 = [1, 2, 3]
>>> lista2 = lista1
```

Na Memória

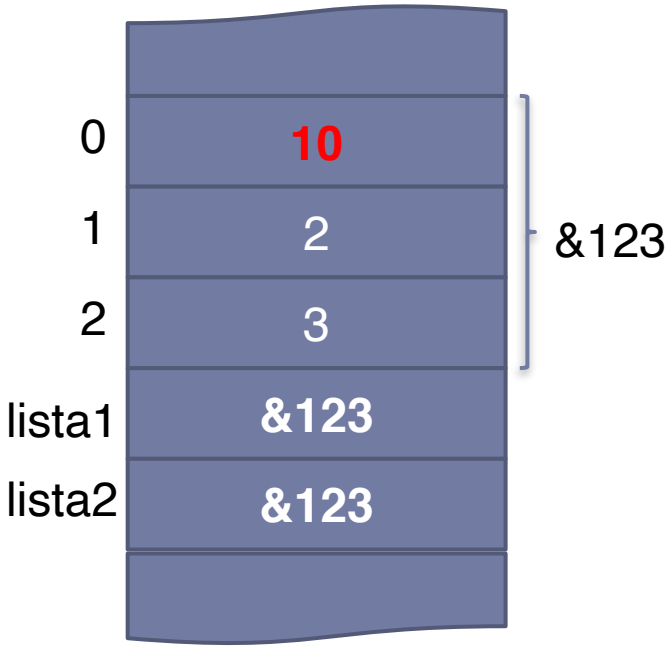


Cópia de Listas

Em Phyton

```
>>> liata1 = [1, 2, 3]
>>> lista2 = lista1
>>> lista1[0] = 10
>>> lista1
[10, 2, 3]
>>> lista2
[10, 2, 3]
```

Na Memória

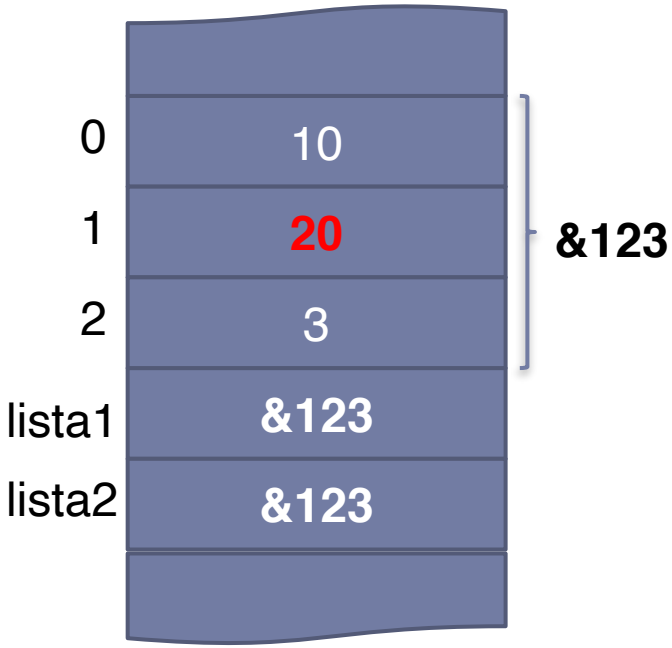


Cópia de Listas

Em Phyton

```
>>> liata1 = [1, 2, 3]
>>> lista2 = lista1
>>> lista1[0] = 10
>>> lista1
[10, 2, 3]
>>> lista2
[10, 2, 3]
>>> lista2[1] = 20
>>> lista2
[10, 20, 3]
>>> lista1
[10, 20, 3]
```

Na Memória



Como evitar isso?

- Duas opções:
  1. Usar um for para copiar valor a valor
  2. Usar slices para fazer a cópia

Opção 1: uso de **for**

```
>>> lista1 = [1, 2, 3]
>>> lista2 = []
>>> for i in range(len(lista1)):
...     lista2.append(lista1[i])
...
```

Opção 2: uso de ***slice***

```
>>> lista1 = [1, 2, 3]
>>> lista2 = lista1[:]
```

Em ambos os casos...

- Alterações em uma lista não são refletidas na outra

```
>>> lista1 = [1, 2, 3, 4, 5]
>>> lista2 = lista1[:]
>>> lista2[0] = 10
>>> lista1
[1, 2, 3, 4, 5]
>>> lista2
[10, 2, 3, 4, 5]
>>> lista1[3] = 20
>>> lista2
[10, 2, 3, 4, 5]
```

Exercícios

1. Faça um programa que percorre uma lista com o seguinte formato: [['Brasil', 'Italia', [10, 9]], ['Brasil', 'Espanha', [5, 7]], ['Italia', 'Espanha', [7,8]]]. Essa lista indica o número de faltas que cada time fez em cada jogo. Na lista acima, no jogo entre Brasil e Itália, o Brasil fez 10 faltas e a Itália fez 9. O programa deve imprimir na tela:

(a) o total de faltas do campeonato

(b) o time que fez mais faltas

(c) o time que fez menos faltas

Exercícios

2. Faça um programa que simule um lançamento de dados. Lance o dado 10 vezes e armazene os resultados em um vetor. Depois, monte um outro vetor contendo quantas vezes cada valor foi obtido. Imprima os dois vetores. Use uma função para gerar números aleatórios, simulando os lançamentos dos dados.

Exemplo de uma possível saída:
[3, 1, 5, 3, 5, 4, 5, 5, 3, 6]
[1, 0, 3, 1, 4, 1]

Exercícios

3. Faça um programa que percorre um vetor e imprime na tela o valor mais próximo da média dos valores do vetor. Exemplo:

vetor = [2.5, 7.5, 10.0, 4.0]

(média = 6.0)

Valor mais próximo da média = 7.5

Exercícios

4. Faça um programa que percorre duas listas e intercala os elementos de ambas, formando uma terceira lista. A terceira lista deve começar pelo primeiro elemento da lista menor.

Exemplo:

lista1 = [1, 2, 3, 4]

lista2 = [10, 20, 30, 40, 50, 60]

lista_intercalada = [1, 10, 2, 20, 3, 30, 4, 40, 50, 60]

Referências

- Slides baseados nos cursos de Aline Paes e Vanessa Braganholo

Manipulação de Vetores

Leonardo Murta
leomurta@ic.uff.br